UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO Y DESARROLLO DE UN SISTEMA DE GESTIÓN PARA CLÍNICAS VETERINARIAS

PABLO IGNACIO GÓMEZ MARTÍNEZ

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

PROFESOR GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
AGUSTÍN VILLENA MOYA

SANTIAGO DE CHILE
2019

DISEÑO Y DESARROLLO DE UN SISTEMA DE GESTIÓN PARA CLÍNICAS VETERINARIAS

Las clínicas veterinarias son un tipo de empresa interesante, ya que su negocio no es tan simple como dar un solo servicio o dedicarse solo a la venta de productos, sino que son una combinación de ambos.

Los procesos de negocio de una veterinaria son principalmente la atención médica de mascotas (como por ejemplo consultas, exámenes, hospitalización y cirugías) y la venta de productos relacionados con animales (como por ejemplo comida para animales, juguetes y accesorios) pero también pueden abarcar otros servicios como peluquería, baño, servicio de transporte, farmacia, atención a domicilio y hotelería.

Hay veterinarias que usan servicios web para mostrar información, llegar a más clientes y ofrecer nuevas formas de interactuar con los clientes. Los servicios web que una veterinaria puede usar son la creación de páginas web informativas, páginas de tienda online y páginas de reserva de horas de atención. Si una veterinaria quiere tener los tres tipos de servicio entonces tiene que elegir y gestionar tres servicios diferentes.

En vista de lo anterior se busca desarrollar un sistema web que pueda cubrir las tres necesidades mencionadas. El sistema propuesto es una solución genérica que pueda servir a múltiples veterinarias al mismo tiempo.

El desarrollo consistió de: diseño del sistema, definición de la arquitectura, diseño de la interfaz web y desarrollo de backend y frontend para lograr las funcionalidades de la solución.

Se describió el diseño e implementación de la solución y se evaluó la solución mediante un formulario de retroalimentación.

Agradecimientos

A mi familia, sin ellos no sería quien soy ni tendría lo que tengo.

A Karol, por su apoyo y cariño.

A Jaime y Nicolás, por acompañarme cuando más lo necesité.

Tabla de contenido

1. Introducción

1.1 Contexto y problema

Las clínicas veterinarias pueden utilizar servicios web para cubrir las siguientes necesidades:

1) Tener visibilidad en internet y así llegar a más clientes. La página web puede aparecer en buscadores y ser compartida entre clientes y conocidos.
2) Comercio electrónico para vender productos y ofrecer otra forma de venta a los clientes.
3) Reserva de horas clínicas para que los clientes puedan reservar con una interfaz web en lugar de solo por teléfono o correo electrónico.

Actualmente existen servicios web y soluciones para cada uno de los tres puntos antes mencionados pero no existe un servicio que cubra las tres necesidades en un solo sistema.

Por lo anterior mencionado, el administrador de una veterinaria se enfrenta a una compleja situación a la hora de querer mejorar sus procesos en alguno de estos tres aspectos, ya que para cada uno debe elegir entre invertir el tiempo y dinero necesarios para investigar y concretar dichas soluciones o seguir operando sin las potenciales mejoras que pueden ofrecer las alternativas tecnológicas. Más aún, las alternativas en cada aspecto son de un nicho específico (solo página web informativa, solo reservas o solo tienda en línea) y es probable que incorporar una combinación de ellas sea complicado debido a que esto requeriría comprar y gestionar varias soluciones al mismo tiempo.

1.2 Solución escogida

Como solución escogida se propone una aplicación web que incluya los tres aspectos mencionados en el contexto (tener visibilidad, comercio electrónico y agenda de horas médicas).

La aplicación web será una sola para todas las veterinarias y sus respectivos clientes.

Una posibilidad es implementar una solución local en cada veterinaria, si bien esto facilita la separación del sistema de cada veterinaria, sería inconveniente a la hora de agregar nuevas veterinarias y también para actualizar el código de cada una cuando hayan cambios, mejoras y correcciones.

El modelo de distribución de software como servicio (SaaS: Software as a Service) es una forma simple para la veterinaria de adquirir la solución y además facilita la distribución de actualizaciones a todas las veterinarias al mismo tiempo. Con solo registrarse la veterinaria puede comenzar a usar el sistema sin necesidad de que se configure una infraestructura nueva (servidor, base de datos, hosting, etc).

Algunas de las funcionalidades pueden conllevar un esfuerzo logístico de parte de la veterinaria, por ejemplo la venta con pagos en línea requiere de un proceso de certificación con Transbank, hay comisiones de por medio y el tema de incluir o no un sistema de despacho con una empresa externa. Es por esto que es conveniente identificar este tipo de funcionalidades y permitir que el administrador de la veterinaria las pueda activar o desactivar con flexibilidad. De esta forma la veterinaria no se verá obligada a incorporar procesos si no está en condiciones de hacerlo todavía.

El módulo de comercio electrónico es desarrollado desde cero en lugar de usar un plugin o sistema existente. Si bien ya existen varias alternativas de comercio electrónico (como por ejemplo Prestashop [9]), estas son sus propios sistemas que no funcionarían directamente con el backend que se construirá para las demás funcionalidades. Esto produciría que la página de comercio electrónico esté separada del resto y no se puedan relacionar las configuraciones, personalizaciones y relaciones de objetos del backend con la parte de comercio electrónico.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar un sistema web para que administradores de clínicas veterinarias puedan tener un sitio web con tienda en línea y reserva en línea.

1.3.2 Objetivos específicos

1. Desarrollar un módulo de configuración de sitio web: que el administrador de veterinaria pueda definir su sitio web sin necesidad de tener conocimientos de programación. La personalización debe ser sencilla y con pocos pasos (definir un nombre, un logo y contenido).
2. Desarrollar un módulo de comercio electrónico con formas de pago en línea que permita al administrador de veterinaria agregar, modificar y eliminar productos del catálogo, ver compras realizadas por clientes y actualizar estado de los pedidos.
3. Desarrollar un módulo de agenda de horas médicas en línea que permita al administrador de veterinaria configurar los horarios disponibles y visualizar horas reservadas.
4. Validar si la solución resuelve el problema mediante una encuesta a usuarios relacionados con veterinarias. A partir de los resultados evaluar si es necesario cambiar el enfoque del proyecto en cuanto a desarrollo y objetivos.

1.3.3 Alcance

El trabajo realizado corresponde a:

- Diseño y desarrollo de un sistema web que le permita a administradores de veterinarias configurar un sitio web informativo, tienda en línea y reservas.
- Validación de la solución mediante un formulario de retroalimentación.

El enfoque del trabajo realizado fue el desarrollo de software y el análisis de la solución y las herramientas utilizadas para lograr las funcionalidades propuestas.

El trabajo realizado no incluye los aspectos comerciales y empresariales asociados a un emprendimiento. Las gestiones y desarrollos necesarios para tener transacciones reales, empresa constituida, emisión de boletas y facturas están fuera del alcance de este trabajo y quedan propuestos como trabajo futuro.

2. Marco conceptual

En esta sección se describen algunos conceptos y tecnologías que fueron relevantes a lo largo del desarrollo de la solución y que podrían considerarse como no tan comúnes en el ámbito del desarrollo web.

REST (Representational State Transfer): Estilo de arquitectura de software. Se usó este estilo para definir un conjunto de operaciones HTTP de interacción entre el frontend y el backend. Estas operaciones no tienen estado y se clasifican como GET (obtener datos), PUT (editar datos) y POST (envío genérico de datos).

CSRF token (Cross-Site Request Forgery): Cada vez que un usuario usa el sistema, se le asigna un CSRF token. El CSRF token es un valor generado aleatoriamente y se requiere cada vez que se da una interacción entre el frontend y el backend como medida de seguridad. Si no se requiere y valida este token, entonces un sitio malicioso distinto podría gatillar una acción del usuario en el sistema sin que el usuario lo quiera y el sistema no podría diferenciar si viene del sitio original o de un sitio malicioso.

HTML DOM (Document Object Model): Objeto que representa el documento HTML de un sitio web. A través del DOM, los elementos de la página pueden ser cambiados dinámicamente.

Bash: Lenguaje de comandos para la línea de comandos de Unix.

JSX: Sintaxis de Javascript que permite definir elementos de interfaz HTML de manera directa en vez de tener que definirlos como strings. Se ocupa comúnmente en proyectos frontend que usan la librería React.

Babel: Transpilador que convierte código moderno de Javascript (Javascript con especificaciones modernas como ES6 y JSX) a Javascript ES5 que es soportado más exploradores (especialmente los más antiguos). Esto permite escribir código Javascript con los últimos avances, beneficios y azúcar sintáctico pero sin perder la compatibilidad con exploradores antiguos.

Webpack: Herramienta que empaqueta código Javascript definido con módulos. Se encarga del proceso de convertir un conjunto de archivos Javascript en un solo archivo en conjunto con Babel y otros plugins. Esta herramienta se usa para que el código en ambiente de desarrollo quede listo para el ambiente de producción.

3. Diseño y Análisis de la solución

En el siguiente capítulo se hará un análisis de la propuesta y se propondrá un diseño de la solución. Se describirán las principales funcionalidades a desarrollar y finalmente se hará un análisis de las tecnologías elegidas para llevar a cabo el desarrollo.

Para tener una idea de qué servicios web utilizan las veterinarias, se realizó un pequeño estudio de mercado. Se buscaron páginas web de veterinarias usando el buscador google.com, amarillas.emol.com y mercantil.com. Para la búsqueda se utilizaron los términos "veterinaria" y "veterinaria santiago". De esta búsqueda se obtuvieron los siguientes datos:

- Se encontraron 31 páginas web de veterinarias.
- Del total, 16 son páginas web informativas sin tienda en línea ni reserva en línea (51.61%), 5 tienen información y tienda en línea (16.12%) y 10 tienen información y reserva en línea (32.25%).
- Ninguna veterinaria incluye tienda en línea y también reserva en línea.
- Solo una página web muestra tienda en línea y reserva en línea (https://www.veterinariahomm.cl) pero la parte de reservas dice "Pronto nuestra reserva de horas ¡Atentos a este mes de junio!" lo cual da la impresión de que el proyecto fue abandonado o tuvo dificultades.
- De las 10 páginas con información y reserva en línea, 6 usan el servicio web "E-reservas" (http://www.zentec.cl/productos/e_reservas) y 4 usan formularios web de contacto.

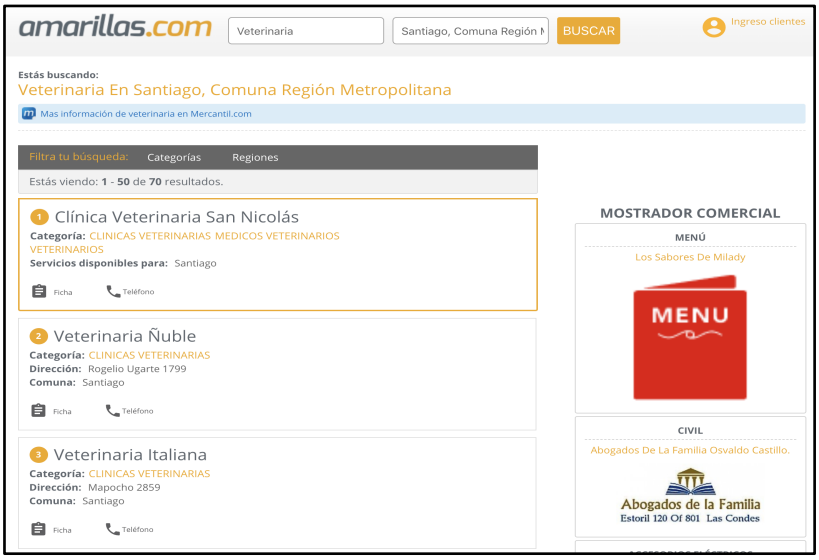

figura 1: búsqueda de veterinarias en internet

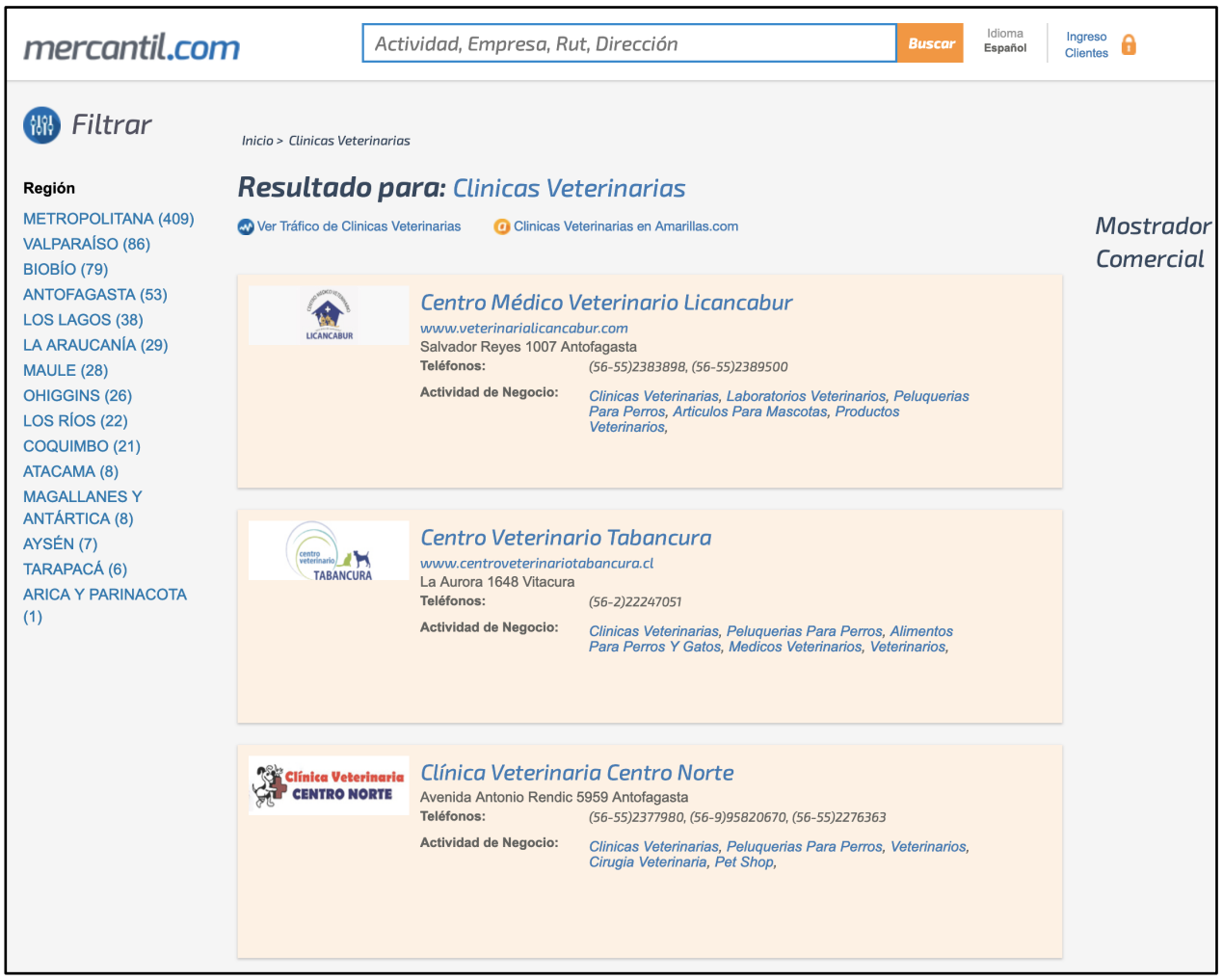

figura 2: búsqueda de veterinarias en internet

Como más de la mitad de las veterinarias encontradas solo usan un servicio web para tener un sitio informativo, esta es la funcionalidad principal que debe tener el sistema. Un sitio informativo le permite a la veterinaria dar a conocer su local en internet y atraer más clientes a su local.

La existencia de veterinarias que usan servicios para tienda en línea y reserva en línea, demuestra que hay algunas veterinarias que están dispuestas a usar soluciones tecnológicas para expandir su negocio en cuanto a cantidad de clientes y formas de interactuar con los clientes.

La idea de la propuesta es cubrir las necesidades tecnológicas de las veterinarias con un servicio que les facilite la creación de un sitio web. La solución se implementará en la forma de una aplicación Web con tres capas: capa de datos, capa de lógica y capa de interfaz de usuario. La capa de datos y la de lógica están en el backend y la capa de interfaz en el frontend. La comunicación entre frontend y backend se realiza mediante una API.

El sistema será distribuido a personas a cargo de veterinarias como SaaS y tanto el administrador como los clientes de su veterinaria podrán acceder a

las funcionalidades mediante un navegador web en computadores y smartphones con acceso a internet.

3.1 Identificación de usuarios de la plataforma

- Administrador: usuario a cargo de la veterinaria, este usuario registra su veterinaria y luego puede definir y configurar su sitio en cuanto a página web, módulo de tienda en línea y módulo de reserva en línea.
- Cliente: representa un cliente de una veterinaria, una vez registrado puede comprar productos en el catálogo y agendar horas médicas.

3.2 Casos de uso

Para los tipos de usuario Administrador y Cliente se proponen los siguientes casos de uso para describir las funcionalidades que tendrá el sistema.

Caso de uso	CU1: Configurar cabecera del sitio
Actores	Administrador
Descripción	Administrador edita el nombre de la veterinaria, subtítulo, correo y número telefónico que se muestran en la cabecera del sitio.

Caso de uso	CU2: Configurar contenido de la sección de inicio
Actores	Administrador
Descripción	Administrador define contenido de la página de inicio, el sistema le permite colocar textos e imágenes con formato como por ejemplo tamaños de letra, letra negrita, colores y centrar párrafos.

Caso de uso	CU3: Administrar catálogo

Actores	Administrador
Descripción	Administrador agrega un producto proporcionando título, descripción y precio. Administrador edita el título, descripción y precio de un producto. Administrador elimina un producto del catálogo.

Caso de uso	CU4: Filtrar productos del catálogo
Actores	Administrador, Cliente
Descripción	Administrador o cliente filtra los productos del catálogo por categoría y escribe en el buscador el nombre de un producto para encontrarlo.

Caso de uso	CU5: Gestionar ventas
Actores	Administrador
Descripción	Administrador ve lista de compras realizadas por clientes y edita el estado de la compra como pendiente o completada según sea el caso.

Caso de uso	CU6: Administrar horario disponible
Actores	Administrador
Descripción	Administrador agrega horas disponibles de atención. Para cada una indica hora, minuto y día de la semana. El sistema guarda la información y luego la muestra en un calendario.

Caso de uso	CU7: Gestión de horas tomadas

Actores	Administrador
Descripción	Administrador ve las horas de atención solicitadas por los clientes. Para cada una el sistema muestra el correo del cliente, la descripción, la fecha de la atención y la fecha de cuándo se realizó la reserva.

Caso de uso	CU8: Comprar productos
Actores	Cliente
Descripción	El cliente compra productos siguiendo la secuencia descrita.
Secuencia normal	1) El cliente agrega productos a un carro de compra. 2) El cliente proporciona información de nombre y domicilio. 3) El cliente paga el valor total de los productos mediante Webpay. 4) El sistema muestra una confirmación de compra exitosa.

Caso de uso	CU9: Agendar hora de atención
Actores	Cliente
Descripción	El sistema muestra un calendario. El cliente puede ver la disponibilidad de horas seleccionando un día en el calendario. El cliente puede agendar una hora seleccionando una hora disponible del día seleccionado y agregando una descripción del motivo de su reserva.

3.3 Arquitectura de la solución

A nivel de arquitectura, a grandes rasgos hay dos alternativas; tener todo en un solo servidor o un servidor para cada veterinaria y uno principal que este conectado al resto.

La solución no es un sistema dedicado a una veterinaria en particular, sino que busca ser un servicio generalizado que pueda servir a múltiples veterinarias. Cada veterinaria a su vez da su servicio a su conjunto de clientes.

Como se trata de un sistema para servir a distintas veterinarias simultáneamente, la solución tiene que ser lo suficientemente general para poder aplicarse a distintas veterinarias.

La primera solución evaluada fue la de un servidor para cada veterinaria (enfoque multi-servidor, figura 3). A continuación se describen ventajas y desventajas de este enfoque.

Ventajas:

- Como cada servidor tiene una base de datos propia, no hay riesgo de tener errores a causa de que una veterinaria afecte a otra.
- Si el servidor de una veterinaria falla, el resto sigue funcionando.

Desventajas:

- Se complejiza la distribución del código cuando haya cambios, mejoras y correcciones.
- Si un usuario es cliente de más de una veterinaria tiene que registrarse más de una vez.
- Cada vez que una nueva veterinaria comienza a usar el sistema tiene que haber un procedimiento de configuración de hosting, dominio, base de datos, control de versiones, instalación del software y despliegue.

A continuación se muestra el diagrama de arquitectura que describe a grandes rasgos la interacción entre los distintos servidores en este enfoque multi-servidor.

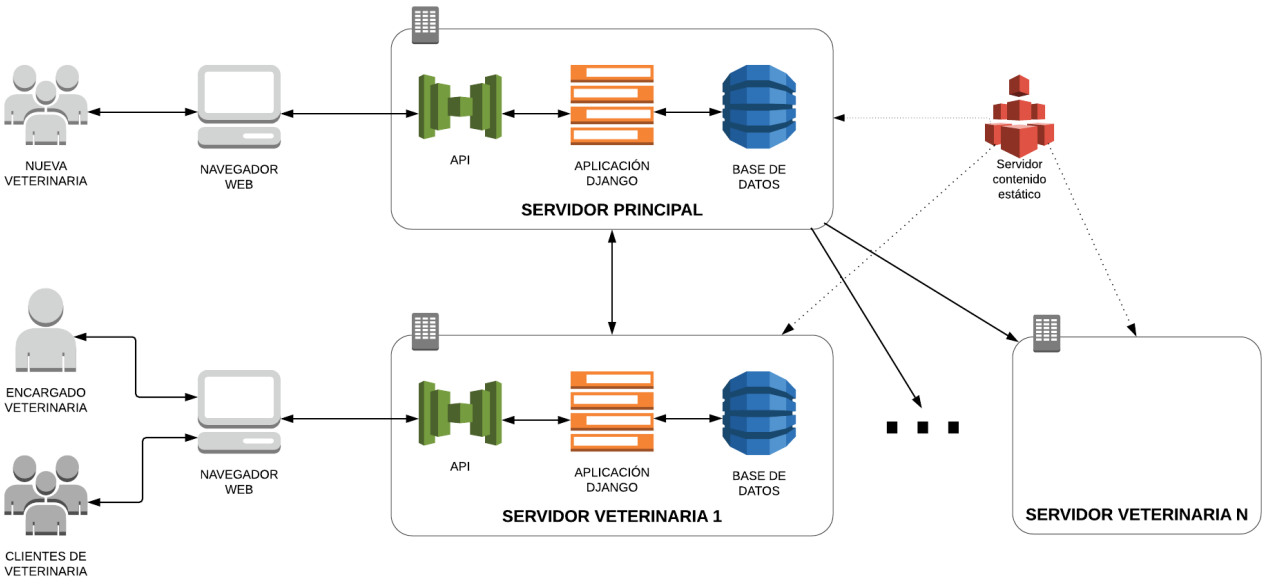

figura 3: diagrama de arquitectura de enfoque multi-servidor

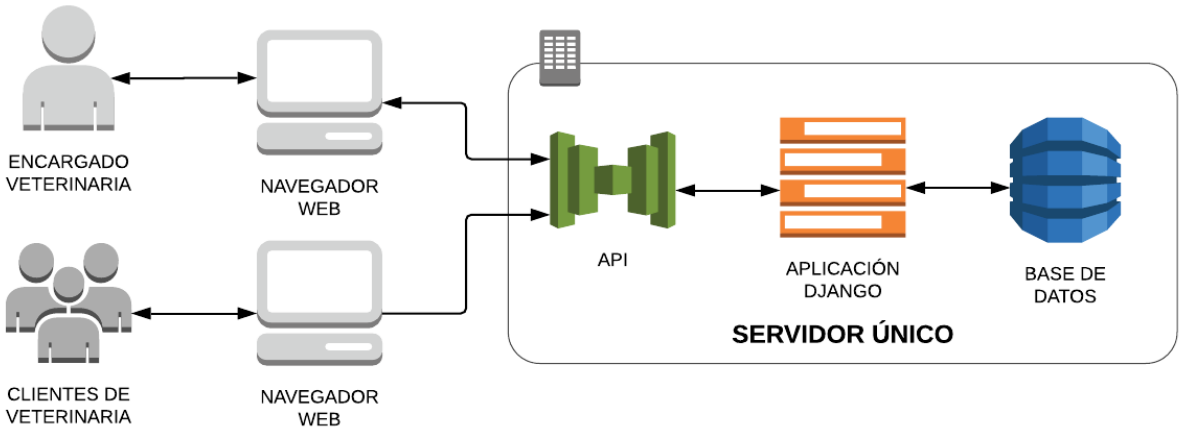

figura 4: diagrama de arquitectura de servidor único

Por otro lado, un servidor único (figura 4) ofrece las siguientes ventajas:

- El código se distribuye a un solo servidor. Esto simplifica las labores de despliegue y actualización de cambios.
- Los usuarios clientes solo tendrán que registrarse una vez y pueden usar su cuenta en cualquier veterinaria del sistema.

- Las nuevas veterinarias que se registren pueden comenzar a usar el sistema rápidamente. No hay necesidad de hacer despliegues de nuevos servidores.

3.4 Modelo de datos

En base a las funcionalidades de la solución propuesta, se definió un modelo de datos que pueda representar la información requerida por el sistema.

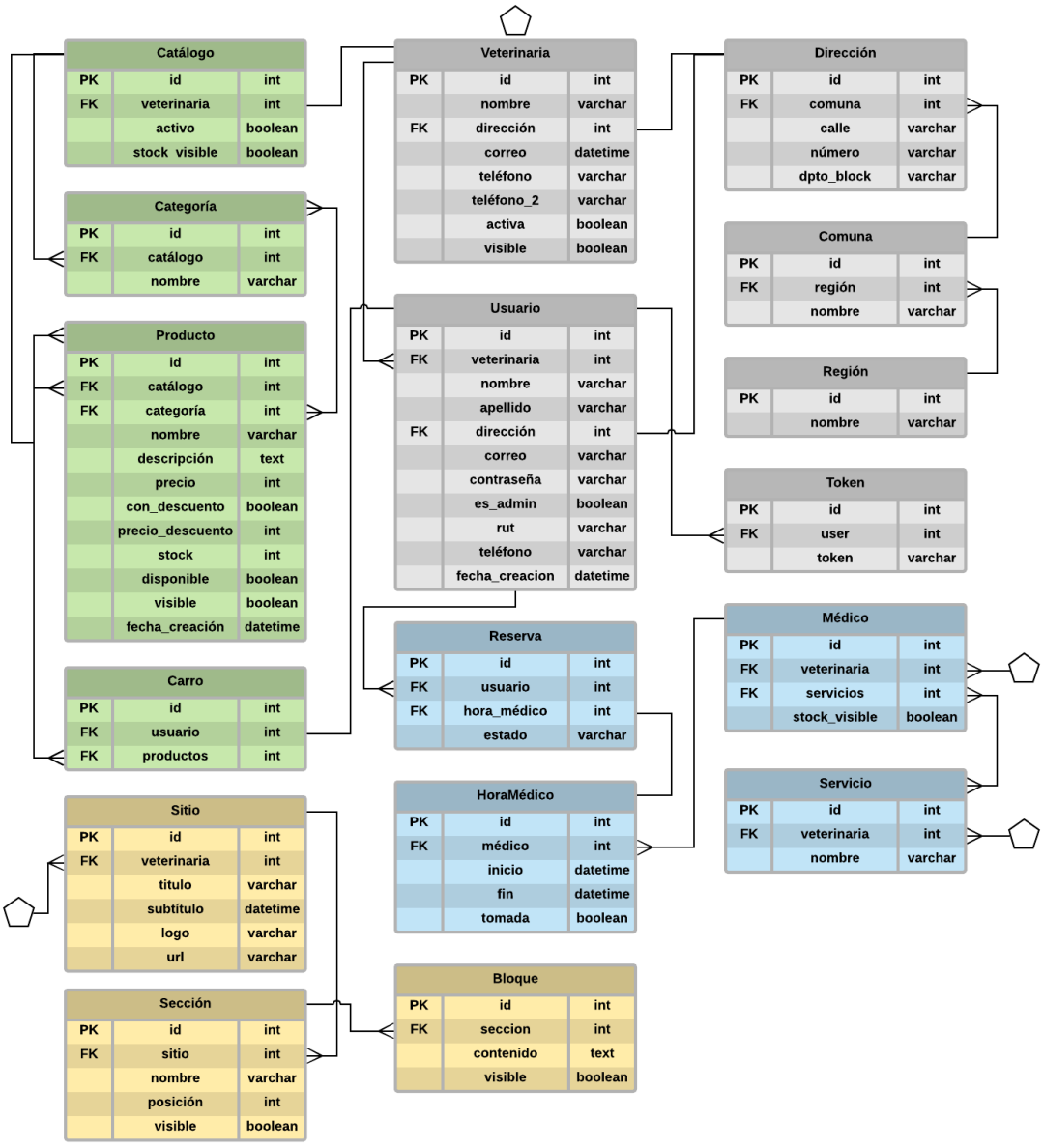

figura 5: diagrama de entidad-relación

En color amarillo se destacan las entidades que están relacionadas principalmente con el módulo de configuración de sitio web. En color verde las relacionadas con el módulo de comercio electrónico y en color azul las relacionadas con el módulo de agenda de horas médicas.

La entidad Usuario es importante ya que permite tener el sistema de autenticación y registro. Un administrador de veterinaria se diferencia de un cliente por el campo es_admin.

La entidad Sitio permite guardar el logo, título y subtítulo de la página de cada veterinaria. Esto es importante para que el administrador de veterinaria pueda personalizar su sitio y los cambios persistan.

La entidad 'Bloque' representa el contenido en una sección de la página web. El campo 'contenido' representa texto con formato, imágenes y enlaces. La definición de estos elementos como texto es posible gracias al uso de WYSIWYG (acrónimo de What You See Is What You Get, en español, "lo que ves es lo que obtienes") [13]. Cuando el encargado agregue contenido a su sitio verá un editor de WYSIWYG que le permitirá definir texto con formato, imágenes y enlaces. Luego el sistema guarda una representación de ese contenido como texto en el campo 'contenido' de la entidad 'Bloque'.

3.5 Mockups de las interfaces

En esta sección se muestran los mockups de las interfaces principales a desarrollar. Estos diseños dan una idea inicial general de los componentes que se desarrollarán.

figura 6: mockup de módulo de configuración de sitio web

El mockup de la figura 6 muestra la interfaz para que el administrador de la veterinaria pueda configurar cómo se ve su sitio web. El administrador podrá editar el contenido de su página de inicio con los textos, imágenes y formatos que estime convenientes.

El mockup de la figura 7 muestra la interfaz para que el administrador de la veterinaria pueda ver las horas disponibles y reservadas.

figura 7: mockup de módulo de reserva de horas

3.6 Tecnologías a utilizar

En esta sección se analizan y describen las tecnologías de programación y desarrollo elegidas para desarrollar las funcionalidades propuestas.

3.6.1 Backend

Se utilizó Django para el desarrollo del backend. Django es un framework web que utiliza el lenguaje de programación Python. Django proporciona un amplio conjunto de herramientas para desarrollar sistemas web, como por ejemplo:

- Sistema para asociar URLs con lógica.
- Definición de modelos con clases.
- Sistema de autenticación.
- Página de administración para agregar, modificar y remover elementos de la base de datos.
- Gran variedad de comandos para gestionar la aplicación web.

La capa de datos consistirá de una base de datos PostgreSQL dado que es más avanzada y ofrece más características que la base de datos SQLite por defecto en Django (PostgreSQL posee más variedad de campos posibles y mejor rendimiento para sistemas complejos [18]).

La conexión entre capa de lógica y capa de datos se realizará mediante Django ORM (acrónimo de Object Related Mapping) con el cual se pueden realizar consultas de base de datos con notación de objetos. Por ejemplo:

Users.objects.filter(active=True)

En vez de

SELECT * FROM Users WHERE active = TRUE

Los modelos son descritos como clases Python con atributos y métodos, es por esto que consultar la base de datos con una notación de objetos resulta ser más natural y directo. Una ventaja de usar ORM es que los cambios a las tablas (por ejemplo agregar una nueva fila) se generan automáticamente con el comando "python manage makemigrations". De esta forma se evitan los errores que pueden ocurrir al hacer migraciones manualmente. En la figura 8 se muestra un ejemplo de una clase Python que representa un modelo en la base de datos.

```python
class ProductCategory(BaseModel):
    name = models.CharField(max_length=50)
    veterinary = models.ForeignKey(
        Veterinary,
        on_delete=models.CASCADE,
        related_name='product_categories'
    )

    class Meta:
        verbose_name_plural = "product categories"

    def __str__(self):
        return self.name
```

figura 8: ejemplo de definición de modelo usando clases

3.6.2 Frontend

En cuanto al frontend (capa de presentación) se usará React. React es una librería de JavaScript para crear y manejar interfaces web. Manejar la interfaz con código de esta forma es conveniente en este proyecto debido a que los tres módulos contienen elementos que cambian dependiendo de la configuración de la veterinaria. Por ejemplo el sitio web debe mostrar secciones y bloques definidos por el encargado y los clientes deben poder ver los cambios configurados por el administrador.

React permite programar interfaces en base a componentes. Con un enfoque de componentes se pueden tomar secciones de una interfaz y dividirlas en componentes independientes y reutilizables. Programar con componentes reduce la duplicación de código porque los componentes pueden definirse de una manera genérica y luego ser reutilizados. Programar

componentes con interfaces y comportamientos específicos y acotados resulta en una alta cohesión del código. Por otro lado los componentes encapsulan todo lo que necesitan (interfaz, datos, funciones y eventos) y esto resulta en un menor acoplamiento de código.

React es una librería que se ha hecho popular en los últimos años [15]. Es una tecnología usada por empresas como Facebook, AirBnb, Uber, Netflix, Instagram, Whatsapp y Dropbox [16].

Django incluye un sistema de plantillas que permiten poner algo de lógica para definir el HTML, CSS y Javascript que se mostrarán en el sitio, pero su lenguaje de marcado no está diseñado para manipular la interfaz cuando hay complejidad y cambios frecuentes (por ejemplo, no es posible definir funciones ni responder a eventos). Es por esto que tener un framework dedicado a la manipulación de la interfaz por el lado del cliente es lo más adecuado para el enfoque del proyecto.

3.6.3 Estilos

Se evaluaron distintas alternativas de estilos para tener los componentes HTML con CSS necesarios para desarrollar funcionalidades y darle a la plataforma una apariencia más profesional. Esta es una lista de las alternativas consideradas:

- Bootstrap: es una librería de frontend que proporciona estilos y facilita el diseño responsivo de interfaces. **Ventajas:** rápida integración al proyecto, es muy usado en el mundo, hay mucha documentación y fuentes de información. **Desventajas:** uso intensivo de jQuery, la cual es una librería JavaScript para manipulación del DOM. React ya cumple esta función y sería redundante usar jQuery.
- Isomorphic: se puede conseguir una plantilla de React que tenga componentes ya incluidos como por ejemplo la plantilla "Isomorphic". **Ventajas:** se consigue una interfaz avanzada y bien trabajada de manera más directa. **Desventajas:** es necesario aprender y adecuarse a una estructura y estilo de código ya existente, también se limitan las posibilidades por las librerías que vienen con la plantilla.
- Bulma: es un framework CSS que proporciona estilos y facilita el diseño responsivo de interfaces. **Ventajas:** solo usa CSS, al no incluir JavaScript es perfectamente compatible con el frontend React, es simple de usar. **Desventajas:** es menos conocido y por lo tanto no tiene tantas fuentes de información ni tantos componentes avanzados.

Para lograr una interfaz avanzada y completa primero se intentó usar Isomorphic. Pero Isomorphic tiene una estructura muy compleja y tiene mucho código propio. Aprender a usarlo, acostumbrarse al estilo de código y ver cómo usar todas las librerías que incluye tomaría demasiado tiempo. Es por esta razón que se descartó esta alternativa después de probarla.

Finalmente se usó Bulma para los estilos de la plataforma, su simplicidad y buena compatibilidad hicieron que pudiera integrarse al proyecto sin mayores problemas.

3.6.4 Otros

Como entorno de desarrollo se utilizó Pycharm, el cual ofrece herramientas avanzadas para escribir, visualizar y ejecutar código Python.

Para el versionamiento del código se utilizó Git con un repositorio del servicio Bitbucket. Git será la principal herramienta con la cual se distribuirá y actualizará el código en el servidores de manera efectiva y segura. Para facilitar el uso de Git se usó el programa GitKraken para visualizar y realizar acciones en el control de versiones.

Los servidores son instancias Amazon EC2 (instancias de computación escalable en la nube) con sistema operativo Ubuntu 16.04. Se eligió Amazon Web Services por ser una empresa reconocida que ofrece productos de buena calidad y variedad. La instancia del servidor es de tipo "T2 micro" con 1 GiB de memoria RAM y rendimiento de red "De bajo a moderado". Si en un futuro se observan problemas de rendimiento, la instancia puede mejorarse.

La configuración de los dominios se realizará en la página www.nic.cl por ser la entidad encargada de los dominios ".cl". Los dominios se enlazarán al servidor principal mediante el servicio de DNS CloudFlare.

El sistema de pagos se implementará con Transbank Webpay ya que es el único de proveedor de pagos con bancos del país. Hay otros servicios como QVO y MercadoPago pero todos ellos son basados en Transbank y por lo tanto tienen la misma funcionalidad pero con tarifas adicionales.

4. Implementación de la solución

En este capítulo se describe la implementación de la solución, los problemas que se encontraron en el desarrollo y las soluciones aplicadas.

4.1 Estructura del proyecto

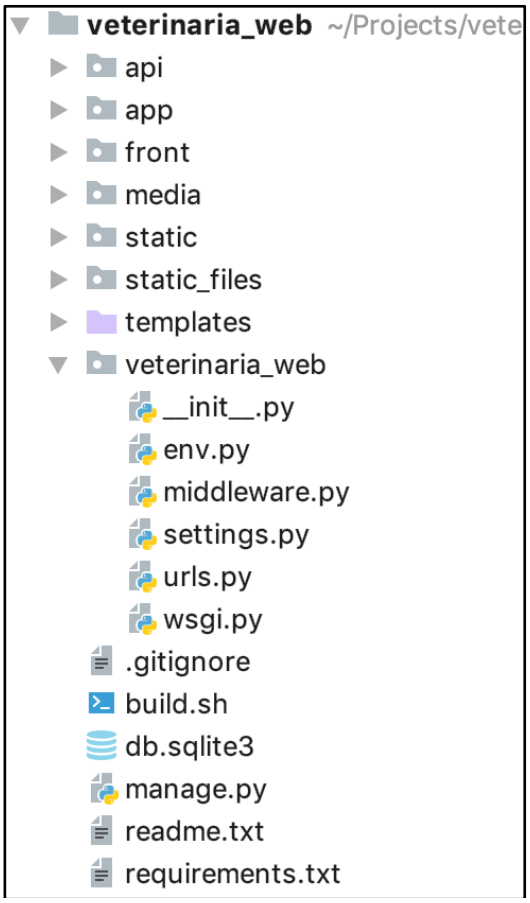

figura 9: estructura general del código del proyecto

El código del proyecto está dividido en carpetas según la funcionalidad que cumplen. En la figura 9 se ve la estructura general del proyecto. En la siguiente lista se describen con mayor detalle las partes que componen el sistema.

- veterinaria_web (figura 10): Carpeta que contiene las configuraciones generales del proyecto como por ejemplo librerías instaladas, parámetros de idioma y formatos y ubicación de archivos estáticos. También contiene la configuración de URL principales del sistema y un

archivo con variables de ambiente que permite diferenciar el ambiente de desarrollo del ambiente de producción.

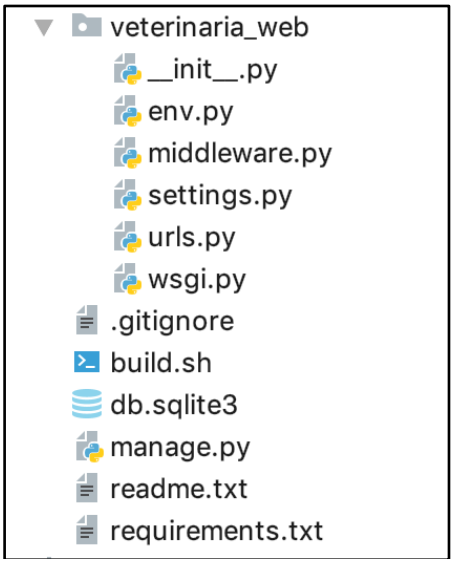

figura 10: carpeta veterinaria_web

- app: Carpeta que contiene los modelos de datos. También tiene la función que maneja el envío de la página inicial al cliente (el resto de las rutas es manejada por el frontend con React).
- api: Carpeta que contiene las URL, la lógica y los serializadores de la API. Los serializadores son classes Python con las cuales se validan los datos enviados desde el frontend y se le da formato a la respuesta JSON enviada desde el backend.
- front (figura 11): Carpeta con el código de frontend separado del backend. El archivo webpack.config.js contiene configuraciones como plugins usados y proxy. El archivo package.json contiene una descripción de las dependencias necesarias para correr el proyecto. Las interfaces y sus componentes están definidos en archivos Javascript dentro de la carpeta Components. Los componentes están divididos en las secciones Auth, Home, Ecommerce y Reservation.

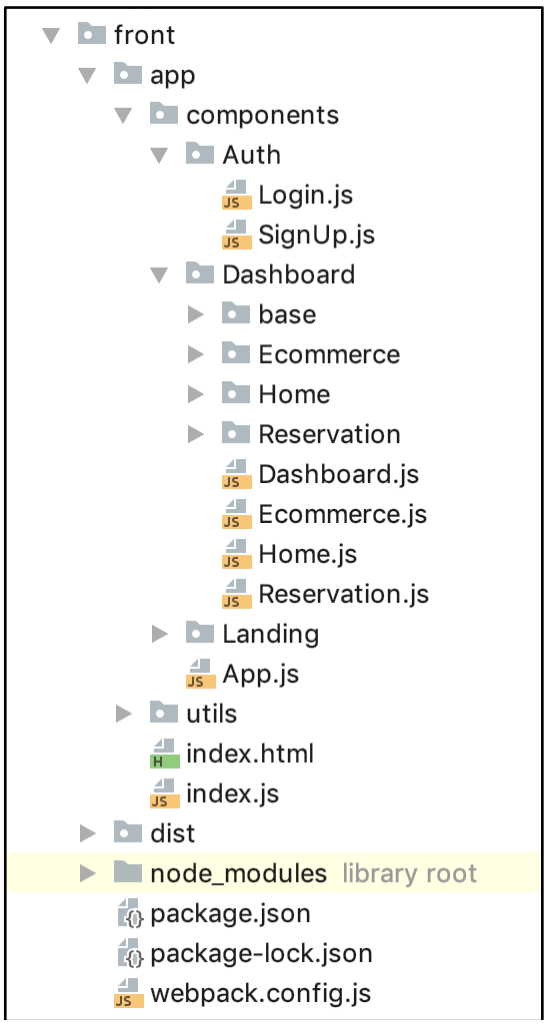

figura 11: estructura del código frontend

- requirements.txt: Archivo que especifica las dependencias Python del proyecto. Es una buena práctica usar este archivo ya que hace que instalar las dependencias sea sencillo con solo un comando.
- build.sh: Script de bash (línea de comandos Unix) que contiene los comandos necesarios para pasar el código del frontend a un solo archivo minificado Javascript. Este procedimiento resulta en un archivo más liviano que carga más rápido. Además de ser más liviano, el archivo se hace más compatible con exploradores web gracias al uso de Babel. Babel es un compilador que toma código escrito con Javascript moderno (jsx, ES6).

4.2 Peticiones API entre frontend y backend

A medida que se van necesitando datos en la interfaz, se realizan consultas HTTP al backend. Las peticiones son autenticadas por sesión. El backend utiliza la herramienta Django REST Framework para manejar las peticiones y responder con datos en formato JSON.

En ambiente de producción backend y frontend funcionan juntos bajo un mismo URL y puerto pero en ambiente de desarrollo el backend funciona en localhost:8000 y el frontend en localhost:8080. Esta diferencia de puertos hace que las peticiones API arrojen error 404 (not found). La solución fue usar un proxy en la configuración de *webpack* para peticiones API y archivos estáticos. La figura 12 muestra esta configuración. Solo es necesario configurar esas dos rutas ya que el resto del sistema es manejado desde el frontend.

```
devServer: {
  proxy: {
    '/api': 'http://127.0.0.1:8000',
    '/static': 'http://127.0.0.1:8000'
  },
  historyApiFallback: true,
},
```

figura 12: configuración de puertos para proxy

Adicionalmente la variable *historyApiFallback* permite que todas las direcciones de URL vayan primero a la URL base y desde ahí la ruta la maneje el frontend (de lo contrario ir a la URL base y navegar funcionaría correctamente pero acceder a la URL directamente no).

También por motivos de seguridad Django no permite peticiones HTTP *cross-origin.* La solución fue configurar una excepción de middleware (software entre el sistema operativo y la aplicación con reglas para las peticiones HTTP entrantes) para *cors* (Intercambio de Recursos de Origen Cruzado). Esta excepción solo se realiza en ambiente de desarrollo por lo que no significa un riesgo de seguridad en ambiente de producción. La figura 13 muestra la configuración realizada.

```python
def dev_cors_middleware(get_response):

    def middleware(request):
        response = get_response(request)

        response['Access-Control-Allow-Origin'] = 'http://localhost:8080'
        response['Access-Control-Allow-Methods'] = 'GET, POST, PUT, PATCH,
        OPTIONS, DELETE, HEAD'
        response['Access-Control-Allow-Headers'] = 'Content-Type, X-CSRFToken'
        response['Access-Control-Allow-Credentials'] = 'true'
        return response

    return middleware
```

figura 13: configuración para aceptar puerto 8080

La autenticación funciona mediante sesión en el servidor y cookie en el navegador web de cada usuario. Durante el desarrollo surgió un problema de peticiones HTTP GET funcionando bien pero peticiones HTTP POST arrojando error 403 (forbidden). La causa del problema era que las peticiones eran realizadas mediante la función JavaScript *fetch* que no tiene problemas de sesión para peticiones GET pero que no incluye la información de la cookie para peticiones POST. La solución fue incluir la cookie en las peticiones POST (figura 14). Para obtener la Cookie se utilizó la librería JavaScript *js-cookie*.

```javascript
import Cookies from 'js-cookie';

export async function fetchPost(url, body) {
  return await fetch(
    url,
    {
      credentials: 'same-origin',
      method: 'POST',
      headers: {
        'X-CSRFToken': Cookies.get('csrftoken'),
        'Accept': 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(body)
    }
  );
}
```

figura 14: uso de cookies

4.3 Implementación de interfaces mediante componentes

Como fue mencionado en la sección 3.7, el uso de la librería React permite crear interfaces modulares y reutilizables mediante el uso de componentes. Un componente es una clase de JavaScript que contiene la definición de una interfaz HTML y que puede tener estado.

Todos los elementos de interfaz del sistema son un componente o parte de un componente. Como ejemplo de la implementación de un componente, en la siguiente figura se muestra el código JavaScript del componente *FormField*.

```jsx
export class FormField extends React.Component{
  static propTypes = {
    label: PropTypes.string.isRequired,
    placeholder: PropTypes.string.isRequired,
    name: PropTypes.string.isRequired,
    type: PropTypes.string,
    defaultValue: PropTypes.string,
    error: PropTypes.string,
    iconLeft: PropTypes.string,
    onChange: PropTypes.func
  };
  static defaultProps = {
    type: 'text',
    defaultValue: '',
    error: '',
    iconLeft: ''
  };

  render() {
    const {
      label,
      placeholder,
      name,
      type,
      defaultValue,
      error,
      iconLeft,
      onChange
    } = this.props;

    return (
      <div className='field'>
        <label className='label'>{label}</label>

        <p className={`control ${iconLeft && 'has-icons-left'}`}>
          <input
            name={name}
            type={type}
            placeholder={placeholder}
            className={`input ${error && 'is-danger'}`}
            defaultValue={defaultValue}
            onChange={onChange}
          />

          {iconLeft &&
            <span className='icon is-small is-left'>
              <i className={`fas ${iconLeft}`}/>
            </span>
          }
        </p>

        {error &&
          <p className='help is-danger'>{error}</p>
        }
      </div>
    )
  }
}

```

figura 15: componente FormField

El componente FormField encapsula todo lo necesario para tener un campo de fomulario: campo HTML, tipo de input, un icono opcional, valor por defecto, función de validación y mensaje de error. Este componente es reutilizado en todos los formularios del sistema. La figura 16 muestra un ejemplo de declaración de este componente. El uso de este componente tiene las siguientes ventajas:

- Permite definir campos de formulario de forma declarativa, solo especificando lo necesario.
- Usa menos líneas de código.
- Es más legible a la hora de revisar el código.
- Evita la duplicación de código.
- Corregir un error en el componente base corrige todos los componentes que lo utilizan.

```
1  <FormField
2    label='Título'
3    placeholder='Título'
4    name='title'
5    defaultValue={title}
6    error={errors.title}
7  />
```

figura 16: ejemplo de uso de componente FormField

Una vez inicializado con variables, el componente tiene esta apariencia en la interfaz (figura 17):

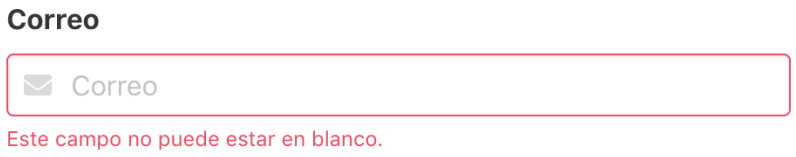

figura 17: ejemplo de interfaz de un componente FormField

Los campos propTypes (figura 15 línea 2) y defaultProps (figura 15 línea 12) permiten definir qué variables de input se espera de un FormField y qué valores deberían tener por defecto. Esto permite prever errores al explicitar las variables requeridas y sus tipos, de lo contrario habría que ejecutar el proyecto contínuamente solo para darse cuenta de este tipo de errores.

Los valores en el HTML están vinculados a las variables Javascript. Cuando éstas variables cambian, la interfaz es automáticamente actualizada. Este concepto es conocido como *UI data binding* y es posible gracias al virtual DOM de React. El virtual DOM es una representación del documento HTML y actualiza el DOM cada vez que detecta cambios.

En el HTML es importante destacar el atributo *onChange*. Este atributo recibe una función lambda y la función sera invocada cada vez que ocurra el evento *onChange*, es decir, cada vez que el usuario modifique el texto del campo HMTL. Esto permite al componente reaccionar a eventos de la interfaz y en este contexto tiene principalmente dos ventajas; mantener el estado de los componentes actualizados con lo que se muestra en la interfaz y poder reaccionar a los cambios de datos (como por ejemplo validar que lo que se está ingresando es correcto).

4.4 Interfaces

figura 18: interfaz de página inicial

Página inicial (figura 18): muestra los servicios que ofrece el sistema y un botón para iniciar sesión y uno para crear una cuenta. Cabe señalar que esta vista está pensada para administradores de veterinaria y no para los clientes de veterinaria. Los clientes de veterinaria accederan directamente al sitio particular de cada veterinaria.

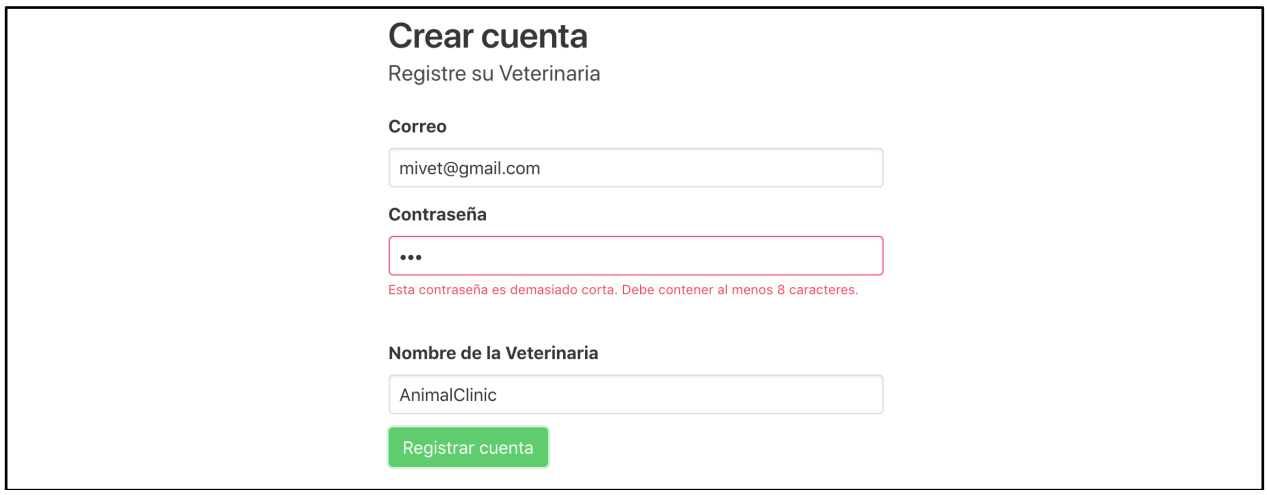

figura 19: interfaz de creación de cuenta

Crear cuenta (figura 19): muestra formulario de creación de cuenta de veterinaria. El campo correo es validado por el lado del cliente por ser un campo HTML input con type "email"[17]. Al clickear el botón "Registrar cuenta" o presionar la tecla "Enter" mientras se esté seleccionando un input, se realiza una llamada de API con método POST y al servidor. Todas las peticiones envían datos de manera segura por el protocolo HTTPS. Los datos se validan una vez más por el lado del servidor (correo válido, contraseña válida y nombre no vacío y único). Si la validación de los datos es correcta, entonces la cuenta es creada y el usuario ingresa inmediatamente al dashboard. Si la validación no es correcta entonces se muestra bajo cada input del formulario un mensaje de error con una descripción del motivo por el cual ese campo no es válido.

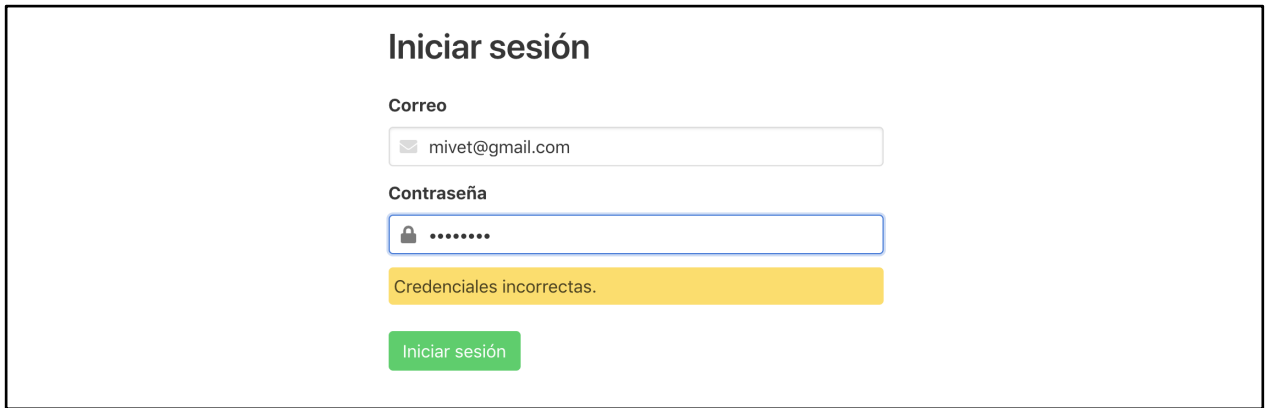

figura 20: interfaz de iniciar sesión

Iniciar sesión (figura 20): al igual que el formulario de registro, el campo de correo es validado. Al llenar los campos y enviar el formulario el servidor revisa si el correo y contraseña están correctos. Si están correctos entonces el usuario es dirigido al dashboard. Si los datos no están correctos se muestra un mensaje de "Credenciales incorrectas.".

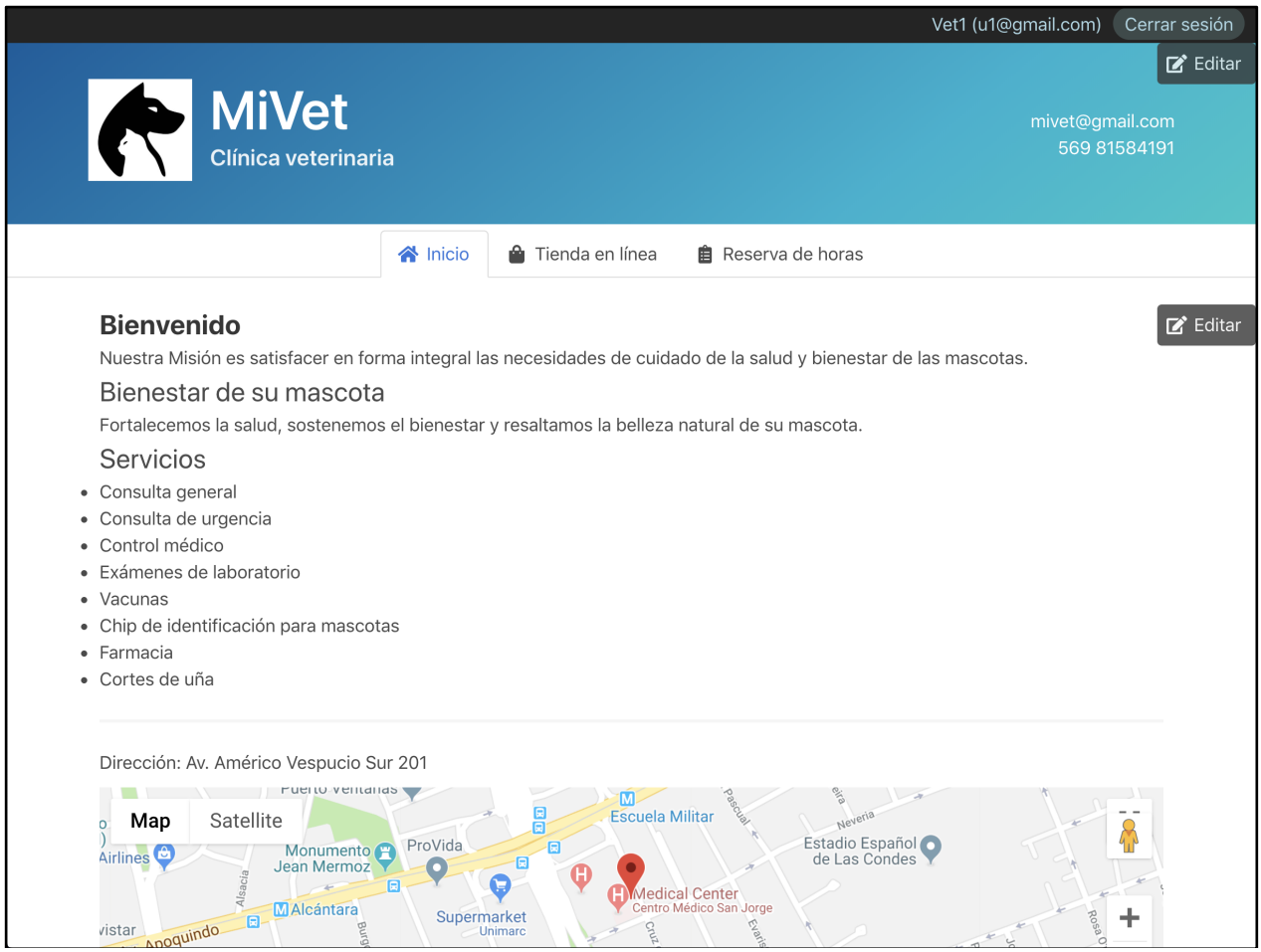

figura 21: interfaz de dashboard

Dashboard (figura 21): página que ve el usuario administrador de veterinaria al iniciar sesión. El administrador puede cerrar sesión con el botón de "Cerrar sesión" en la parte superior derecha de la página. En la cabecera (sección superior con fondo de color) el botón "Editar" permite cambiar los datos que aparecen en la cabecera. En el cuerpo de la página se ve la información que la veterinaria le muestra a sus clientes. Aquí es donde la veterinaria puede poner detalles sobre los servicios y productos que presta, su ubicación y horario, entre otras cosas que la veterinaria quiera mostrar.

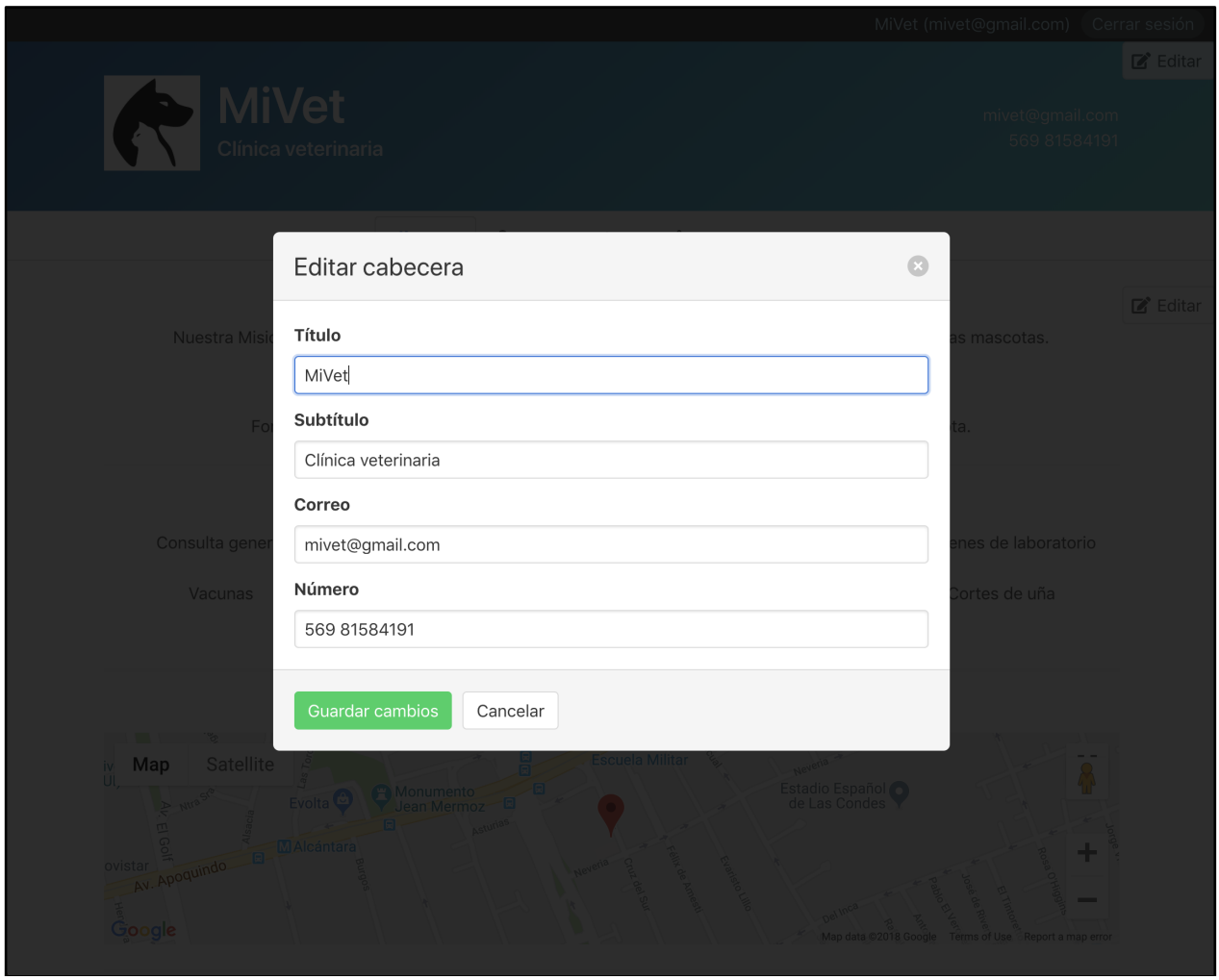

figura 22: interfaz de editar cabecera

Editar cabecera (figura 22): al hacer click en el botón “Editar” ubicado en la sección superior derecha de la cabecera de la página, aparece un modal desde el cual se pueden modificar los datos que aparecen en la cabecera. Los cambios se envían al backend mediante API y los cambios se ven reflejados inmediatamente en la interfaz sin necesidad de refrescar la página.

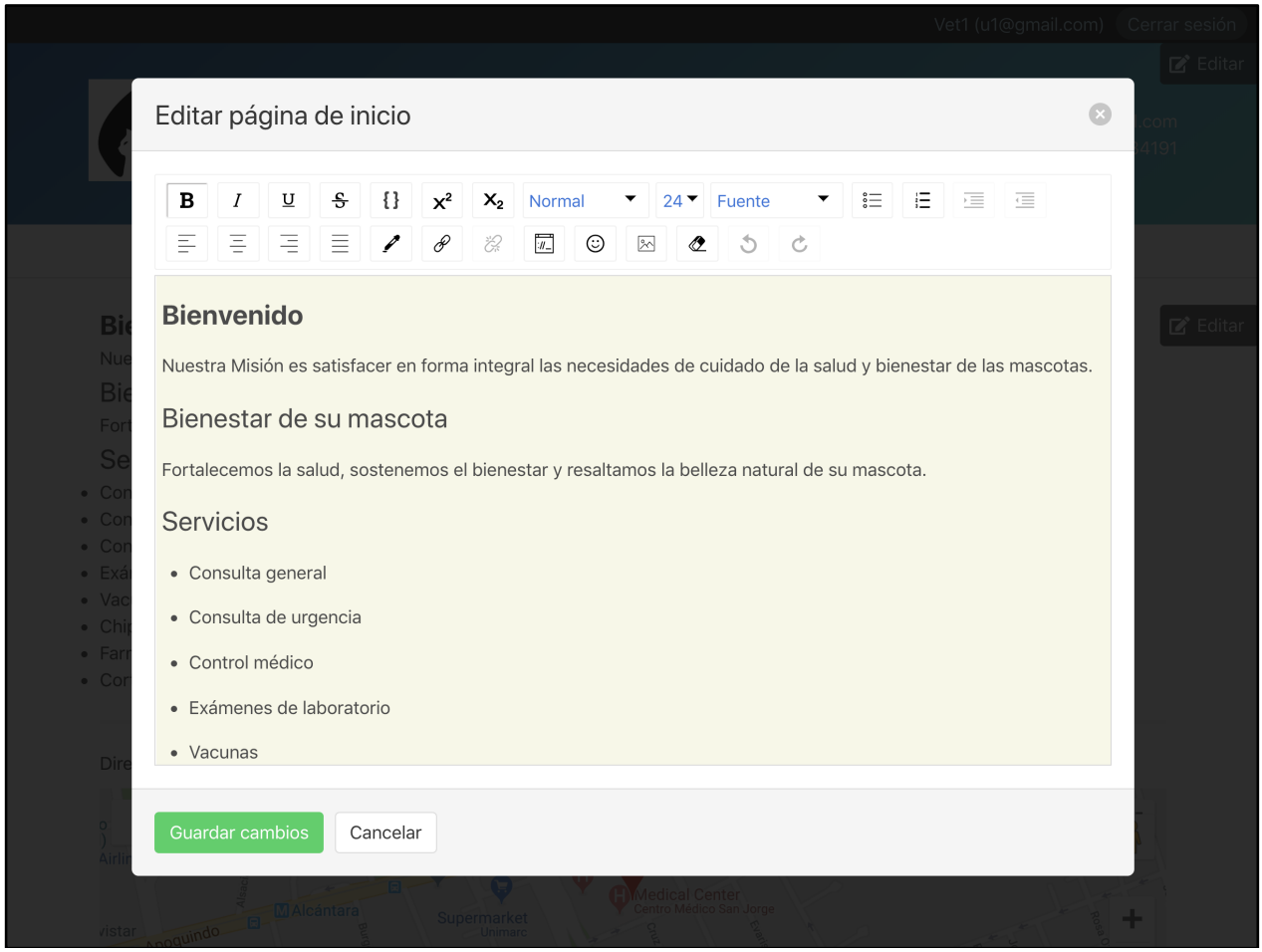

figura 23: interfaz de editar contenido

Editar contenido (figura 23): al hacer click en el botón "Editar" ubicado a la derecha del contenido de la página de inicio, aparece un modal que le permite al administrador configurar el contenido que desea mostrar en la página. La edición ofrece una serie de opciones de formato. Al hacer click en "Guardar cambios" se gatilla una llamada API al backend para guardar los nuevos datos. Una vez guardados el modal se cierra y la página muestra los cambios inmediatamente.

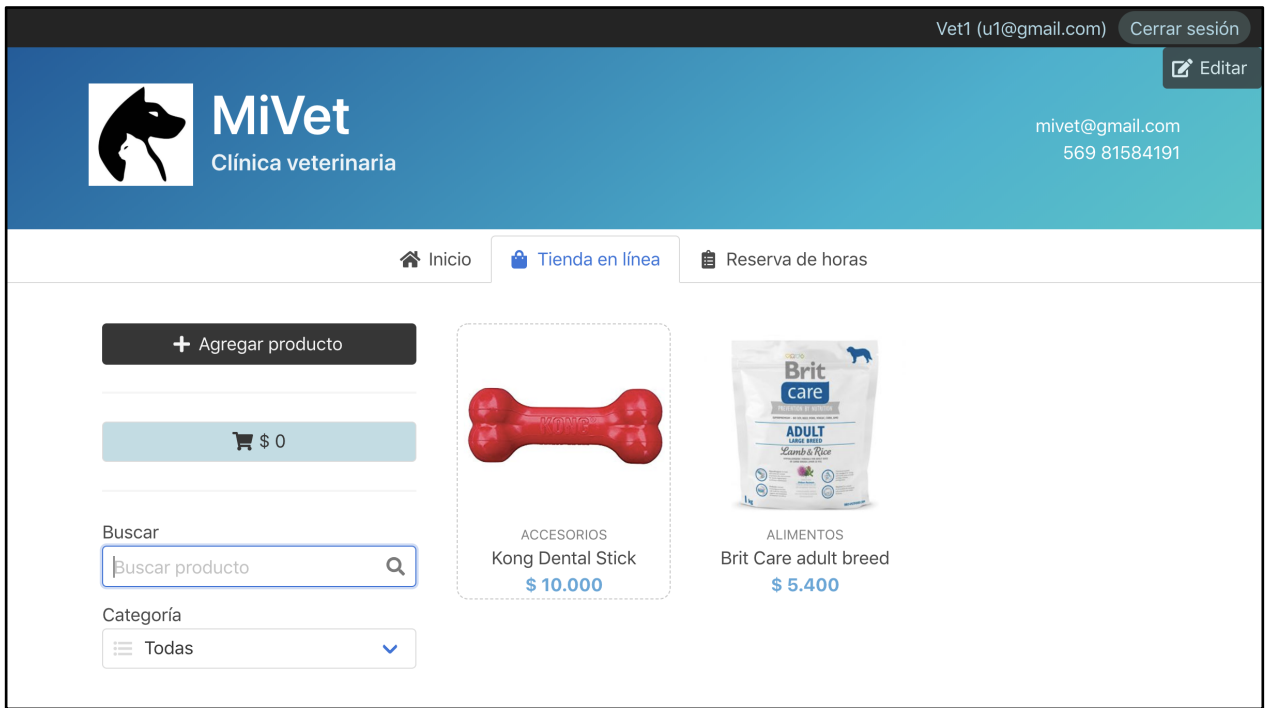

figura 24: interfaz de página inicial

Tienda en línea (figura 24): Se muestran los productos que tiene la veterinaria. La funcionalidad de búsqueda de productos va filtrando los productos inmediatamente a medida que el usuario escribe. Esto puede significar un gran impacto en el rendimiento de la página si el número de productos es muy alto. Cabe señalar que el filtrado es sobre los datos que ya fueron obtenidos con la carga de la página y no se realizan peticiones cada vez que se filtra, lo cual evita consultar excesivamente al servidor. El filtro de búsqueda muestra los productos cuyo nombre contenga el texto que escribió el usuario.

La comparación se hace con minúsculas para no descartar productos que estén escritos igual pero con mayúscula. El filtro de categoría muestra los productos de esa categoría y puede combinarse con la búsqueda por texto para un resultado más acotado.

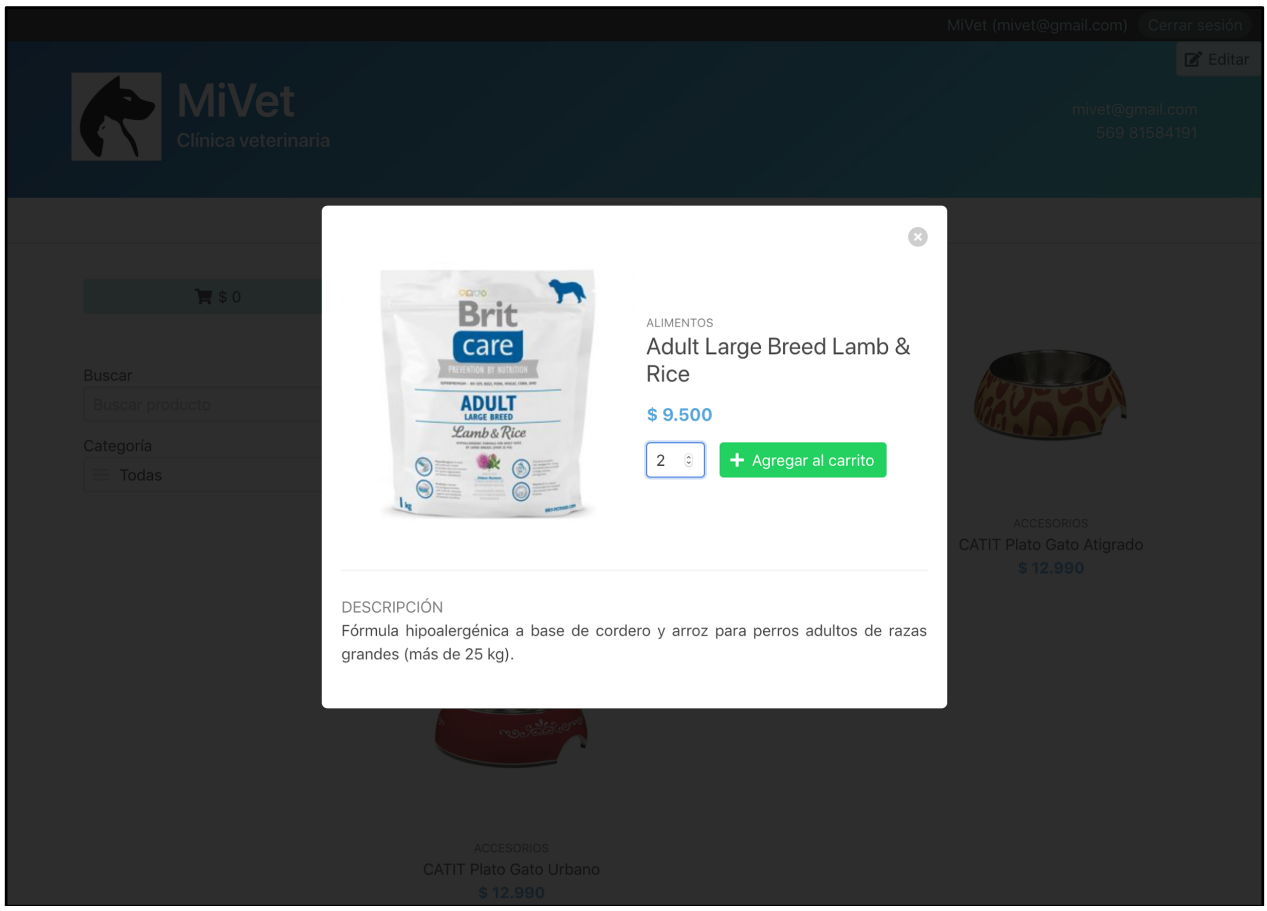

figura 25: interfaz de detalle de producto

Detalle de producto (figura 25): Al hacer click en un producto aparece un modal con los datos del producto (categoría, nombre, precio y descripción).

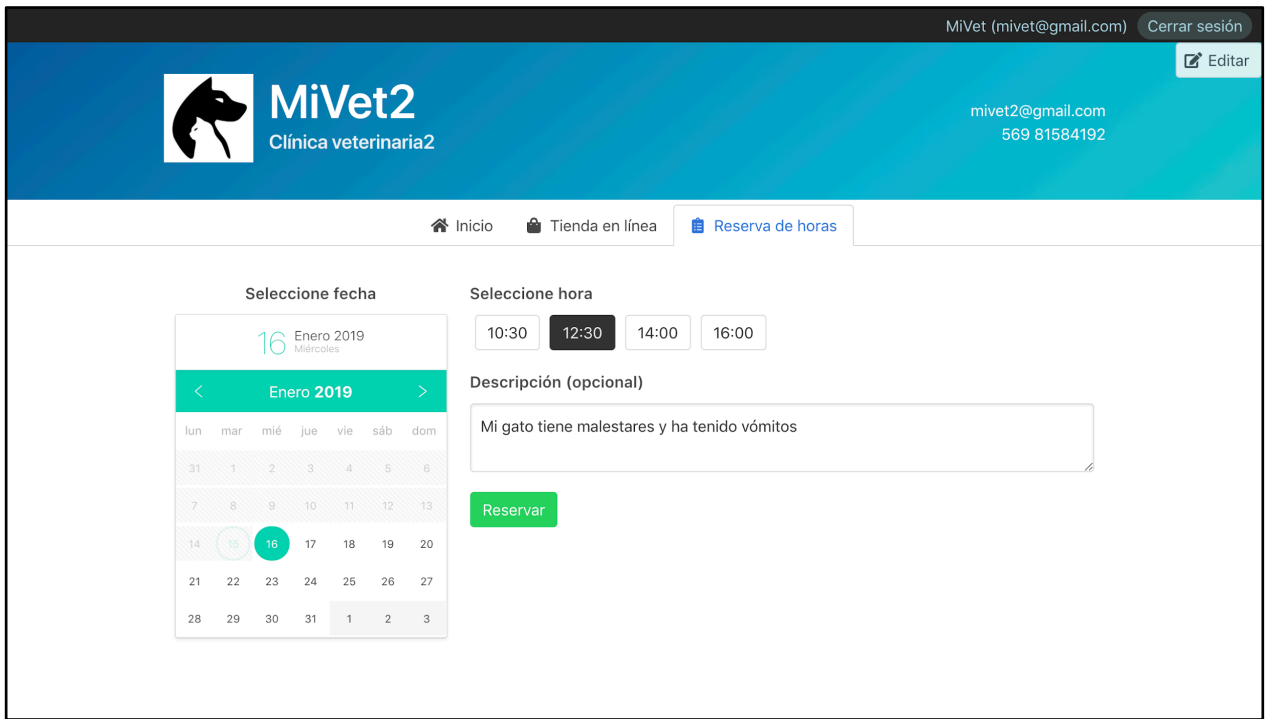

figura 26: interfaz de reserva de horas

Reserva de horas (figura 26): Se muestra un calendario del mes actual. Al seleccionar un día en el calendario, la sección de horas se actualiza con la disponibilidad definida para ese día. Se puede seleccionar una hora, agregar una descripción y presionar el botón "Reservar" para reservar una hora.

En cuanto a las interfaces del cliente, estas son las mismas interfaces que ve el administrador pero excluyendo los botones de edición ("Editar", "Agregar producto", etc). Diseñar las interfaces de esta manera tiene la ventaja de programar solo una vez las interfaces que van a servir tanto para administradores como para clientes. También le permite al administrador tener una visualización inmediata de cómo se verá su página para los clientes.

4.5 Transbank

La documentación de Transbank se ubica en https://www.transbankdevelopers.cl/. Como el backend del proyecto usa el lenguaje de programación Python, el SDK correspondiente es *libwebpay* (https://github.com/TransbankDevelopers/libwebpay-python). Libwebpay está desactualizado ya que usa Python 2 en lugar de Python 3, requiere de un entorno Ubuntu (lo cual impide el desarrollo en sistemas operativos Windows y Mac) y usa librerías modificadas lo cual impide el uso de entornos virtuales.

Para resolver el problema de incompatibilidad con el sistema se analizó la posibilidad de configurar un servidor adicional con el entorno adecuado con el único fin de ser intermediario entre el servidor del sistema y Transbank. La siguiente figura muestra un diagrama de cuáles serían las interacciones necesarias para llevar a cabo un ejemplo de un proceso con el SDK de Transbank.

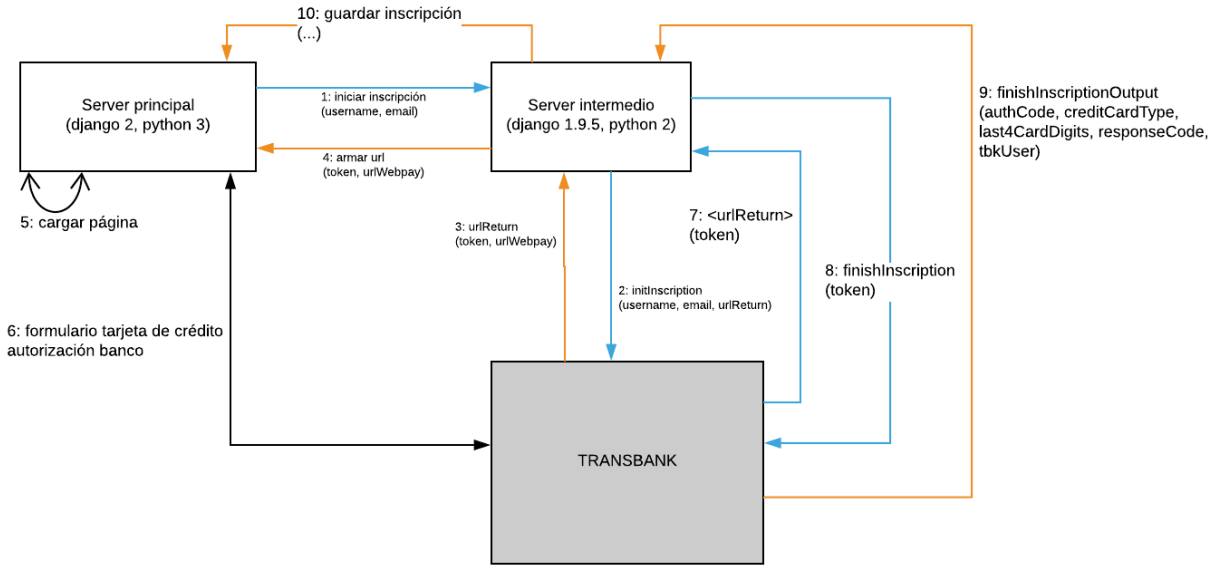

figura 27: diagrama con interacciones entre servidor principal, intermediario y Transbank

Como puede verse en la figura, el tener un servidor intermediario aumenta la cantidad de interacciones API necesarias para hacer cualquier acción con Transbank. Por lo tanto este enfoque complejiza el sistema, lo hace más difícil de implementar y aumenta las posibilidades de fallos futuros.

En el transcurso de la implementación del enfoque mencionado, Transbank renovó su documentación haciéndola más clara. Se usaron los canales de comunicación para obtener ayuda y un encargado de Transbank explicó que están trabajando en el nuevo SDK de Python pero que demorará meses en estar listo. Sin embargo indicó que por mientras se puede utilizar el SDK desarrollado por la empresa Cornershop que también usa Django y Python en su backend. El SDK desarrollado por Cornershop está ubicado en el siguiente repositorio: https://github.com/cornershop/python-tbk. Gracias al SDK de Cornernshop se pudo implementar el cliente Python para interactuar con Transbank. En la siguiente figura se muestra un ejemplo de uso del cliente Python.

```
def webpay_service():
    commerce = Commerce.init_from_files(
        settings.commerce_code,
        settings.key_data,
        settings.cert_data,
        settings.tbk_cert_data,
        INTEGRACION
    )

    return WebpayService(commerce)
```

figura 28: inicialización del cliente webpay

```
webpay = webpay_service()
tbk_response = webpay.init_transaction(
    amount,
    buy_order,
    return_url,
    final_url
)
r = response.result
if r['responseCode'] != 0:
    raise Exception('transbank responseCode ' + str(r['responseCode']))
```

figura 29: iniciar una transacción

A pesar de que se logró la implementación técnica del cliente Python para las funcionalidades de transbank en ambiente de pruebas, la validación con Transbank para pasar a producción es un proceso largo y demoroso. Para llegar a ocupar Transbank en ambiente producción se requieren una serie de pasos y requisitos:

- Iniciación de actividades o crear una empresa.
- Afiliarse con Transbank (subir documentos y que sean aprobados).
- Implementar todas las funcionalidades de Webpay.
- Corregir errores y problemas con la implementación acudiendo a canales de ayuda de Transbank.
- Realizar pruebas de las funcionalidades y casos requeridos por los protocolos de Transbank.
- Documentar resultado de las pruebas y completar formularios de evidencia de integración.
- Enviar documentos y según las respuestas de parte de Transbank realizar correcciones.
- Cambiar certificados y realizar pruebas nuevamente.

Es por esta razón que la funcionalidad de compras con cobros y transacciones reales está fuera del alcance de este trabajo y queda como trabajo futuro.

5. Validación

5.1 Procedimiento

Para validar si las veterinarias tienen las necesidades que se plantearon y que la solución propuesta resuelve el problema, se realizó una encuesta de retroalimentación.

Las preguntas del formulario de retroalimentación tienen como objetivo:

1) Identificar a quién responde la encuesta para saber si es alguien que trabaja en una veterinaria, si es un médico veterinario o un cliente de veterinaria.
2) Determinar si el sistema satisface las necesidades de una veterinaria en general.
3) Para los módulos de sitio web informativo, tienda en línea y reservas, determinar si es necesario dicho módulo y si el módulo es bueno.
4) Obtener una calificación global del sistema (excelente, bueno, regular o deficiente).
5) Proporcionar un campo de observaciones para los comentarios que tengan los usuarios.

El formulario fue distribuido por los siguientes medios: foro institucional de la Universidad de Chile, red social Facebook, correo electrónico a veterinarias. Junto con el formulario se proporcionó un link al sitio web del sistema para que las personas pudieran probar el sistema antes de contestar.

5.2 Preguntas

A continuación se muestra el formulario de retroalimentación con sus preguntas tal como se presentó a las personas encuestadas.

¿Cuál es su relación con veterinarias? *

- ◯ Trabajo o he trabajado en una veterinaria
- ◯ Soy un médico veterinario o estudiante de medicina veterinaria
- ◯ Soy o he sido cliente de una veterinaria
- ◯ Otro: _____

figura 30: formulario de retroalimentación, pregunta 1

*

	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo
El sistema satisface las necesidades de una veterinaria	◯	◯	◯	◯	◯
Poder configurar una página de inicio es útil/necesario	◯	◯	◯	◯	◯
La configuración de página de inicio es buena	◯	◯	◯	◯	◯
Poder configurar una tienda en línea es útil/necesario	◯	◯	◯	◯	◯
La configuración de tienda en línea es buena	◯	◯	◯	◯	◯
Poder configurar y ver reservas es útil/necesario	◯	◯	◯	◯	◯
La configuración de reservas es buena	◯	◯	◯	◯	◯

figura 31: formulario de retroalimentación, pregunta 2

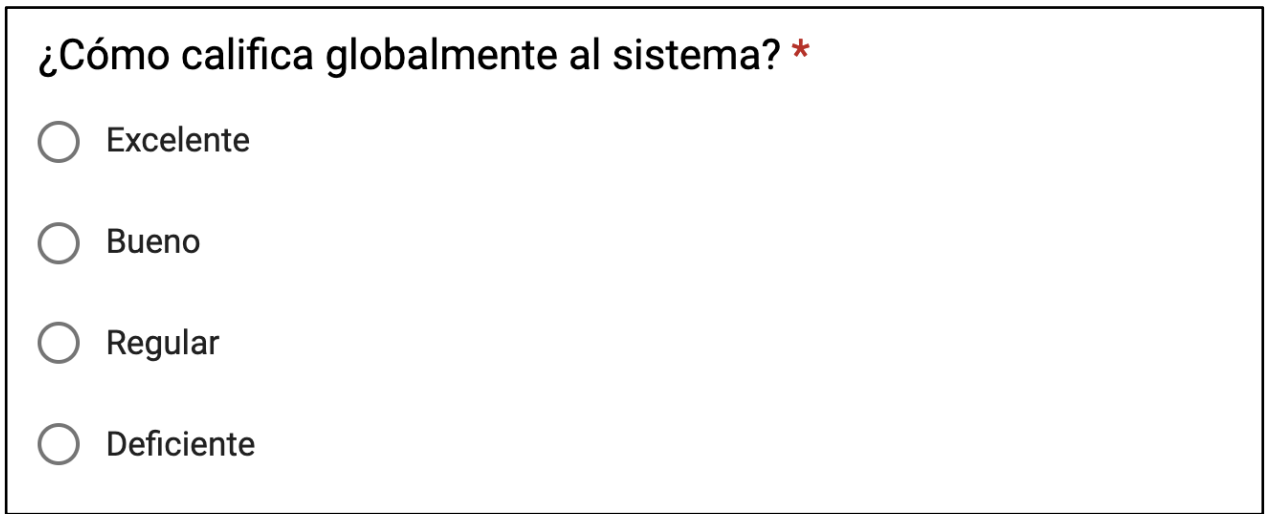

figura 32: formulario de retroalimentación, pregunta 3

figura 33: formulario de retroalimentación, pregunta 4

5.3 Respuestas

A continuación se muestran las respuestas obtenidas.

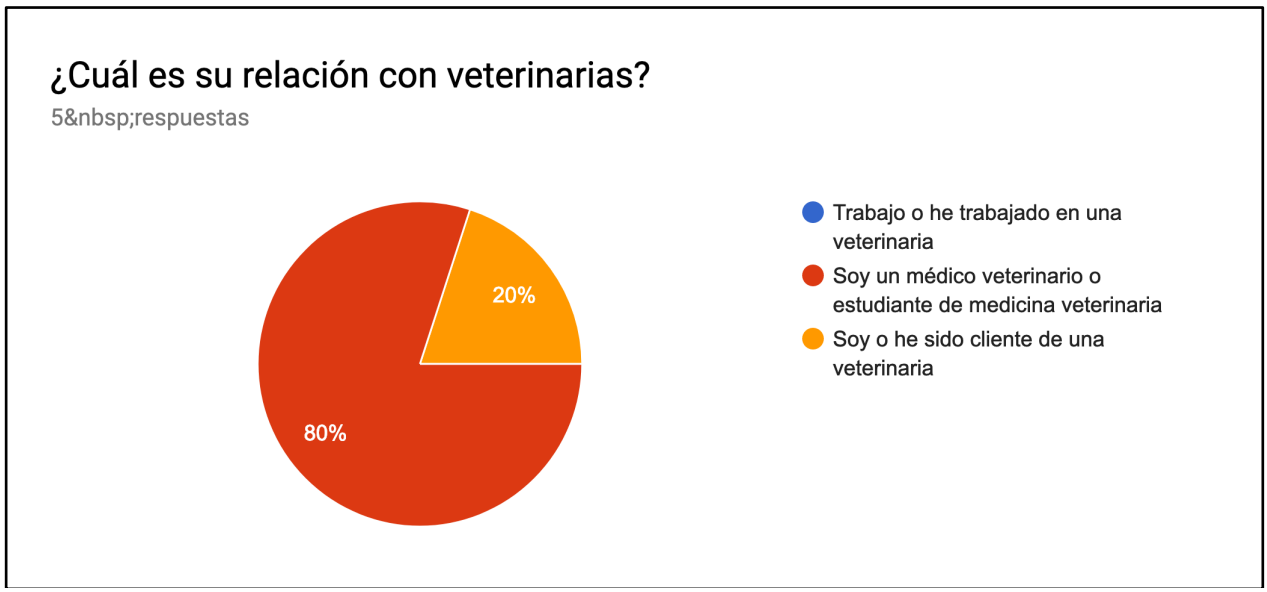

figura 34: respuestas del formulario, pregunta 1

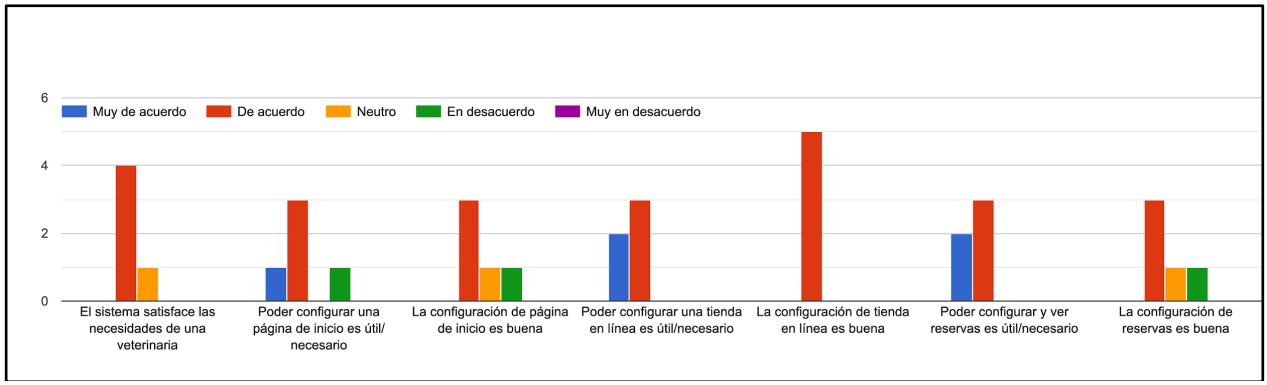

figura 35: respuestas del formulario, pregunta 2

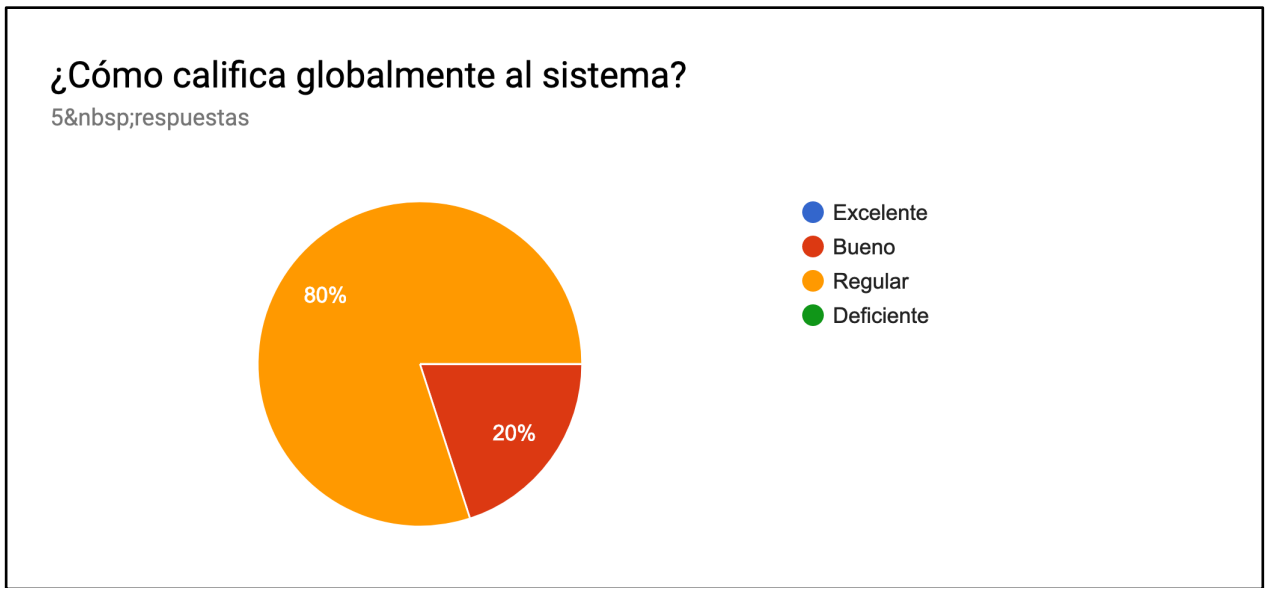

figura 36: respuestas del formulario, pregunta 3

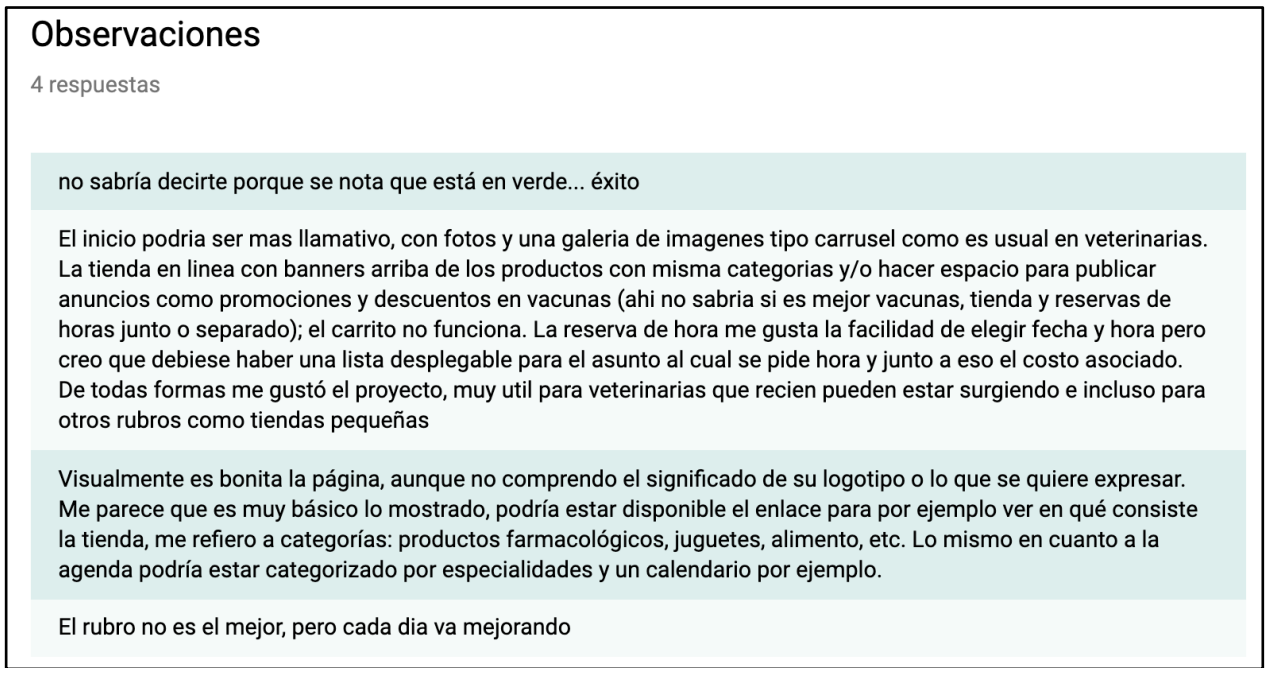

figura 37: respuestas del formulario, pregunta 4

5.4 Análisis

La mayoría de las respuestas fueron de parte de médicos veterinarios o estudiantes de medicina veterinaria. No se obtuvieron respuestas de parte de trabajadores o administradores de veterinaria, esto pudo haber ocurrido por los medios de difusión elegidos o por no haber interés en responder de parte de ellos.

La pregunta general "El sistema satisface las necesidades de una veterinaria" tuvo la mayoría de las respuestas "De acuerdo". En cuánto a las preguntas que buscan ver si es útil/necesario tener cada uno de los módulos (configuración de sitio web, tienda en línea y reservas) la mayoría de las respuestas es "De acuerdo". Por lo tanto el objetivo del proyecto de cubrir estas necesidades se mantiene.

En las respuestas de si las configuraciones de los módulos son buenas, las respuestas tienden a ser "De acuerdo" pero el módulo de configuración de inicio y el módulo de reservas recibieron una respuesta "Neutro" y una respuesta "En desacuerdo". Estos módulos pueden ser mejorados y pulidos en un futuro pero se requeriría de una retroalimentación más específica para identificar estas oportunidades de mejora.

Para la pregunta de calificación global del sistema, el 80% de las respuestas fue "Regular" y el 20% "Bueno". De esto puede interpretarse que

en general el sistema otorga los servicios necesarios de forma suficiente pero que actualmente no posee nada que lo destaque como una solución extraordinaria.

De las observaciones se pueden extraer las siguientes conclusiones:

- Es usual en varias veterinarias tener fotos y galería de imágenes con formato de carrusel. Esta es una funcionalidad que podría ser agregada al sistema dado que las veterinarias estarían acostumbradas a tenerlo y otorga más información a los clientes.
- Se señala que podría agregarse una selección del tipo de asunto de la reserva. Esta es una posible funcionalidad futura pero que actualmente puede cubrirse esa necesidad colocando el asunto en la descripción de la solicitud.
- Al ingresar por primera vez, la tienda y los horarios de reservas se ven vacíos. Si hubiese información inicial de ejemplo (como en la página de inicio y la cabecera), esto ayudaría al administrador a comprender más rápidamente el sistema y tener una idea de cómo quedaría después de configurarlo.

6. Conclusiones

Se diseñó un sistema web para que las veterinarias tengan una alternativa web de servicios unificados. Se desarrolló un módulo de sitio web informativo con cabecera y contenido configurable, un módulo de tienda en línea con creación de productos y filtro de resultados y un módulo de reservas con selección de día y hora y descripción.

Se evaluó la solución mediante un formulario de retroalimentación. La recepción de los usuarios fue generalmente positiva pero hay oportunidades de mejora en cuanto a potenciales funcionalidades (como por ejemplo carrusel de imágenes para página de inicio y reservas con asunto y médico específico).

Se utilizó una cantidad considerable de tiempo de desarrollo en el aprendizaje de la tecnología React ya que se trata de un paradigma de programación frontend distinto a los tradicionales templates incluidos en Django. Utilizar tecnologías nuevas o diferentes puede ofrecer ventajas de rendimiento y buenas estructuras de código pero esto conlleva un mayor tiempo de desarrollo dedicado al aprendizaje, la experimentación y la corrección de errores.

Se utilizó una cantidad considerable de tiempo también en investigar y probar la solución de pagos Transbank Webpay. En medio del desarrollo surgieron dificultades para usar Transbank con el backend de Python por no haber herramientas actualizadas y por requerir gestiones comerciales y empresariales fuera del alcance del proyecto. En retrospectiva, haber priorizado el desarrollo de otras funcionalidades hubiese sido mejor para el proyecto. Sin embargo, cuando a futuro se hagan las gestiones necesarias, la investigación y las pruebas realizadas servirán para concretar la implementación de pagos.

El trabajo tuvo como enfoque el diseño de una solución, el desarrollo de software y el análisis de la solución. Sin embargo, para el funcionamiento real del proyecto se requieren gestiones propias de un emprendimiento. Por lo tanto se propone como trabajo futuro las gestiones comerciales necesarias para transformar la solución desarrollada en un negocio. Las gestiones necesarias incluyen la constitución de una empresa, iniciación de actividades, generación de facturas electrónicas, afiliación con Transbank (con sus requisitos correspondientes), marketing y difusión, etc.

7. Bibliografía

[1] *El desregulado auge de las clínicas veterinarias. El Mercurio, 17 de mayo de 2017.* http://impresa.elmercurio.com/Pages/NewsDetail.aspx?dt=13-05-2017%200:00:00&NewsID=487644&dtB=31-01-2017%200:00:00&BodyID=3&PaginaId=12 (Obtenido: 25 de abil de 2018).

[2] *Hospital veterinario Mota Mascota* https://www.facebook.com/motamascota (Obtenido: 25 de abril de 2018).

[3] *Centro veterinario SaludAnimal Chile* https://www.facebook.com/SaludAnimalChile (Obtenido: 25 de abril de 2018).

[4] *Hospital Veterinario Trinidad* https://www.facebook.com/hospvetetrinidad (Obtenido: 30 de agosto de 2018).

[5] *Amarillas.com.* https://amarillas.emol.com/clinicas-veterinarias/santiago (Obtenido: 30 de agosto de 2018).

[6] *Mercantil.com.* https://www.mercantil.com/clinicas-veterinarias-veterinarias/1896/
(Obtenido: 30 de agosto de 2018).

[7] *PMH, páginas web.* https://www.pmh.cl/paginas-web (Obtenido: 30 de agosto de 2018).

[8] *Sitios web Chile.* http://www.sitioswebchile.cl/ (Obtenido: 30 de agosto de 2018).

[9] *Ventas online crecieron 32,9% el segundo semestre 2017*. 6 de febrero de 2018.
http://www.cnc.cl/ventas-online-crecieron-329-el-segundo-semestre-2017/
(Obtenido: 30 de agosto de 2018).

[10] *Cracker, diseño web ecommerce.* https://www.cracker.cool/diseno-web-ecommerce/ (Obtenido: 30 de agosto de 2018).

[11] *Prestashop.* https://www.prestashop.com/es/software-ecommerce (Obtenido: 30 de agosto de 2018).

[12] e-Reservas, servicio que ofrece un sistema de agenda de horas médicas. http://centralvet.e-reservas.cl/portal/empresas (Obtenido: 30 de agosto de 2018).

[13] Ejemplo de WYSIWYG. https://jpuri.github.io/react-draft-wysiwyg (Obtenido: 9 de septiembre de 2018)

[14] Making React and Django play well together. https://fractalideas.com/blog/making-react-and-django-play-well-together/ (Obtenido: 20 de diciembre de 2018)

[15] React usage. https://2018.stateofjs.com/frontend-frameworks/react/ (Obtenido: 23 de diciembre de 2018)

[16] Sites using React. https://github.com/facebook/react/wiki/Sites-Using-React (Obtenido: 23 de diciembre de 2018)

[17] Especificación del lenguaje HTML para input email. https://html.spec.whatwg.org/multipage/input.html#e-mail-state-(type=email) (Obtenido: 15 de enero de 2019)

[18] SQLite vs PostgreSQL - Which database to use and why? https://tableplus.io/blog/2018/08/sqlite-vs-postgresql-which-database-to-use-and-why.html (Obtenido: 23 de febrero de 2019)